Darwinian selection in a Boolean network

Florian Song

Imperial College London

Introduction

In this poster I will discuss a way to model natural selection as suggested by Charles Darwin.
This model uses a Boolean network with the agents/nodes playing local zero-sum games against each other. The outcomes of these games are determined by the *phenotype/strategy* and the *environment* of the agents. The Darwinian update is then performed by replacing the worst performing agent by a new agent with new properties and the above procedure is subsequently repeated.

Motivation

The model is motivated by natural selection in the sense that certain phenotypes are tested for their fitness in the given environment, formed by other agents. The loser being removed from the system can be thought of as an individual or even a species being submerged by others, because of their unsuccessful strategy. Likewise, the new agent can resemble a mutation within a species which is then on trial for viability being born into the niche of the previously unfit.

Model

This project including the model is largely based on and inspired by the article [1], but small alterations were made as appropriate to render the model more coherent and more natural for implementation.
As mentioned above, the model revolves around a Boolean network composed of N nodes that can attain the two states $S_i = 0$ or $S_i = 1$ for agent i. There are two types of updates, that the network has to undergo:

1. The games (subdivided into two steps):

   (a) A series of N games, where each of the agents acts as an *aggressor* and is assigned to a random *other* agent called the *opponent*. In this model, no two agents are mutual aggressors. The games go as follows: If $S_{agg} = S_{opp}$ then the aggressor scores $+1$ and the opponent -1, otherwise the aggressor scores -1 and the opponent score $+1$.

   (b) One evolution of the network. For this, every agent is also assigned to K *other source* agents. For the remainder of the project, $K = 2$. Now each agent is also associated with a certain Boolean function, that determines the next state of this particular agent. Let the state of agent i at time step t be $S_i(t)$, then

   $$S_i(t) = f_i(S_{i_1}(t-1), S_{i_2}(t-1))$$

   where f_i is a Boolean function and S_{i_1} and S_{i_2} are the two source agents. This Boolean function is weighted with a parameter $P = \frac{3}{4}$, i.e. three of the four possible inputs $(\{0,0\},\{0,1\},\{1,0\},\{1,1\})$ give the same output whereas the last one gives the other output. This gives 8 possible f_i.
   NOTE: This Boolean function is equivalent to the *genotype*. Also, the source agents and the opponents constitute the *environment*. Together, they eventuate in the agents *phenotypic behaviour*.

   For the update, step (a) is carried out first, where all N agents participate on average in two games. Then step (b) is parallelly executed for every agent.
   This is now done repeatedly until an *attractor* is found. Since the state space is finite, there is always going to be a certain constellation at a certain time step that occured before. Because the update is entirely deterministic, the sequence inbetween will be repeated to infinity. This is the *attractor*. The time to reach it is called the *transient*.

2. The Darwinian update

   In this update, the worst performing agent is the agent with the most negative overall score after the games in step (a). This node is then replaced by a new agent, who is associated with a new Boolean function, new source agents and a new oppenent. These are all chosen at random whilst being careful not to create any mutual aggressors.
   NOTE: The connections of other agents involving the loser are not altered.

Implementation

I implemented this model in MATLAB, which might not be the most efficient software package for such a model, but it did a good job overall.
Many of the ideas how to set this up came from the MATLAB RBN Toolbox written by Christian Schwarzer. However, this toolbox was not powerful enough for the model, so that a significant amount was altered or added by me. In particular, an effort was made to vectorise most of the existing functions, since large parts of the toolbox were running on `for` loops. Also, the implementation of the games and the Darwinian update was entirely written by me.

Main Ideas

1. In order to set up the network, a *structure array* called "node" with quite a few fields was used. Some selected fields are:

   node.state The current state i.e. 0 or 1

   node.nextState The next state

   node.input Indices of the *source agents*

   node.rule A 4×1 matrix containing binary entries according to which the next state is determined.

   node.score The score after N games in one time step

   node.opp The index of the opponent

2. A $N \times N$ matrix called "conn" specified the connections between the nodes, i.e. the source agents.

3. A $4 \times N$ matrix called "rule" specified for each node, what the next state is depending on the states of the source agents.

Sample Code

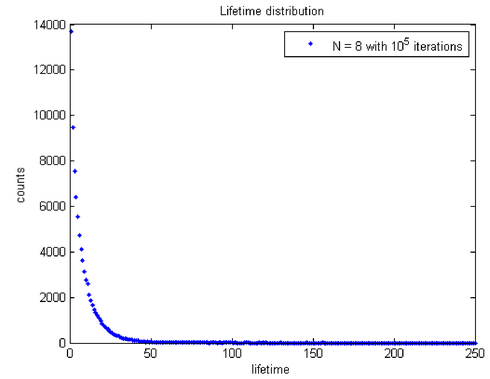
This is some code showing how I implemented the games. However, it is only a very small part of the code.

```
function [node] = Games(node)

for i = 1:length(node)
   if node(i).state == node(node(i).opp).state
      node(i).score = node(i).score + 1;
      node(node(i).opp).score =
      node(node(i).opp).score - 1;
   else
      node(i).score = node(i).score - 1;
      node(node(i).opp).score =
      node(node(i).opp).score + 1;
   end
end
```

Results

The results presented here are without exception drawn from my own calculations. Because my computing power was rather limited and MATLAB is known for its weakness with `for` loops, I was unfortunately not able to get simulations of similar volume. However, by shrinking the size of the network (ie. small N), I was able to achieve numbers of iterations close to the ones in the article.
Whilst the reference article handled a few different aspects of the model, I focused on the number of updates survived by agents at the time when they are replaced in the Darwinian update. This number is called the *lifetime*. Below is a plot produced with my implementation for $N = 8$ and 10^5 iterations of the model:



The main result is here, that this suggests power-law-like behaviour where $counts(l) \sim l^{-0.91}$, where l is the lifetime.

References

[1] **Daniel Eriksson and Henrik Jeldtoft Jensen** *Darwinian Selection in a locally unstable Boolean network* published in the Journal of Statistical Mechanics, 2004.

[2] **Christian Schwarzer (supervised by Christof Teuscher)** MATLAB *RBN Toolbox* Available at: `http://www.teuscher.ch/rbntoolbox/`

[3] **Eyecatcher** from `satisfactorycomics.blogspot.com`